Hierarchical Queues: general description and implementation in MAMBA Image library

Nicolas Beucher, Serge Beucher

Hierarchical Queues: general description and implementation in MAMBA Image library

Nicolas BEUCHER
www.mamba-image.org

Serge BEUCHER
CMM/ARMINES/Mines Paristech

April 2011

1 Introduction

Hierarchical queues (HQ) are extremely efficient structures for fast computation of morphological transformations such as watershed lines or geodesic reconstructions.

The recent development of MAMBA, new morphological image processing library, led the authors of this paper to review the definition and implementation of the watersheds and geodesic reconstructions algorithms based on these hierarchical queues. This library incorporates very fast implementations of these operators, based on HQ. These implementations increase deeply the performances of MAMBA in real-time applications.

Indeed, the documents describing HQ are quite succinct [4, 6]. Unfortunately, these descriptions are far from being accurate and complete. The HQ implementation as it is described in these documents will be used as a starting point for the description of its implementation in the MAMBA library. Regarding the watershed transform, two implementations are available: the first one where only the catchment basins are generated and the second one where these catchment basins are separated by the watershed lines. However, only the first implementation is described in the above mentioned documents. The second one is not addressed although it is more difficult to achieve. The two implementations however are available in some software libraries [7, 8].

The HQ implementation as it is realised in these libraries will be used as a starting point for the description of its implementation in the MAMBA library. Some of its peculiarities and defects will be recalled before indicating which enhancements have been achieved in MAMBA.

Regarding geodesic reconstruction, the description of HQ implementations in the above documents exhibits a peculiar feature which makes this algorithm to be tautological. We shall come back to this characteristic and we shall describe an implementation that avoids this problem.

2 Reminder : Hierarchical Queue general design

Let us start by recalling what hierarchical queues are and how they are used.

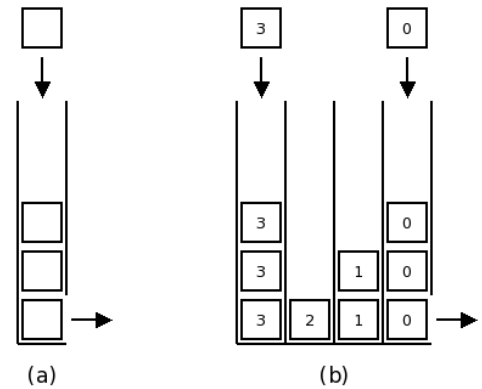


Figure 1: Simple queue (a) and hierarchical queue (b)

A hierarchical queue is the assembly of N simple queues (Figure 1 (a)). A queue is also called FIFO ("First In First Out") register. The queues contain tokens which are introduced and extracted according to a mechanism

which will be described below. In each queue, tokens are extracted in the same chronological order as they were introduced. Each token represents a pixel of the image. The only information carried by a token is the coordinates (x, y) of its corresponding pixel in the image. There is no obligation for a pixel to be represented by only one token. Each queue has a single priority level. This priority level is generally linked to the grey values of the pixels, but it does not necessarily correspond to them directly. By convention, the lower the priority number, the higher the priority of the queue. Therefore, queue 0 has the highest priority (this convention comes from the fact that, in the watershed implementation which was the first HQ-based implementation, pixels of lower grey values are processed first). The number N corresponds to the number of grey levels in the image (256 grey levels in MAMBA library).

All queues are always open at their top: at any time, a token can be inserted in the corresponding priority queue. However, only the highest priority queue can be emptied and cleared: the token which is extracted from the highest priority queue is the one which arrived first in the queue (Figure 1 (b)). Finally, once the highest priority queue is empty, it is removed and the next queue can begin to empty. A lower priority queue cannot be processed if all the higher priority queues are not emptied. Conversely, when a queue is empty and is removed, it cannot be created again.

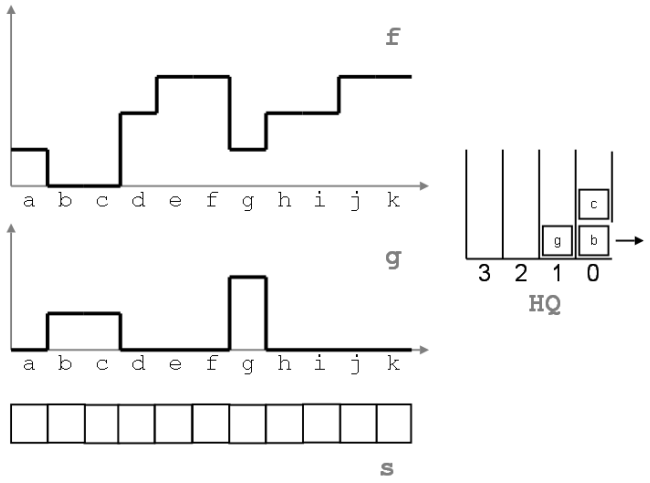


Figure 2: The four components of the structure: original image, label image, status image and HQ

Three images are associated with a HQ (Figure 2). The first one is the image to be treated, **f**. This image defines the number of queues and the tokens priorities. The second image, called result image and denoted **g** both contains the initialisation state of the process and the result of the transformation when achieved. The status image **s** contains various tags (or flags) which control the status and the use of each pixel (and therefore of its corresponding token(s)) of the image.

3 Watershed transformation implementation by hierarchical queues (HQ)

Let us see how this general architecture is used to build watershed transformations (WTS).

3.1 Generating a WTS without catchment basins boundaries

Let us describe first how a WTS is realised with a HQ in the simplest case where only the catchment basins are built. This implementation is available in MAMBA (*basinSegment*).

3.1.1 Initialisation of the process

The image **g** is called label image in the WTS implementation. It is initialized with values corresponding to the labels of the different sources of flooding (minima or markers). Each source of flooding (connected component) is assigned a positive integer value (a label). It is not necessary that the different labels be consecutive values. You may also assign the same label to different connected components. In this case, these connected components will be considered as a single flooding source. Pixels which are not labelled are assigned a zero value in **g**.

The HQ is initialized by storing tokens corresponding to the pixels labelled in image **g** in their respective queues. In the case of the watershed transform (WTS), this priority level of each queue corresponds to the grey level of the token (pixel). The pixels of lower grey level have the highest priority (queues are numbered according to the grey levels, queue 0 has the highest priority). Remind that these tokens carry one single information: the coordinates of the corresponding pixel in **f** or **g**. Therefore, in the sequel, when there is no ambiguity, a pixel or its corresponding token will be given the same naming. The order of introduction of tokens in a queue is arbitrary. It generally corresponds to the image scanning order.

The status image **s** uses two tags:

- "CANDIDATE" indicates that the pixel has not been processed (not pushed on the queue) yet.
- "QUEUED" indicates that the pixel has been queued.

The status image his initialized with all pixels set to "CANDIDATE" except those corresponding to the labelled pixels in g which are set to "QUEUED".

3.1.2 HQ processing

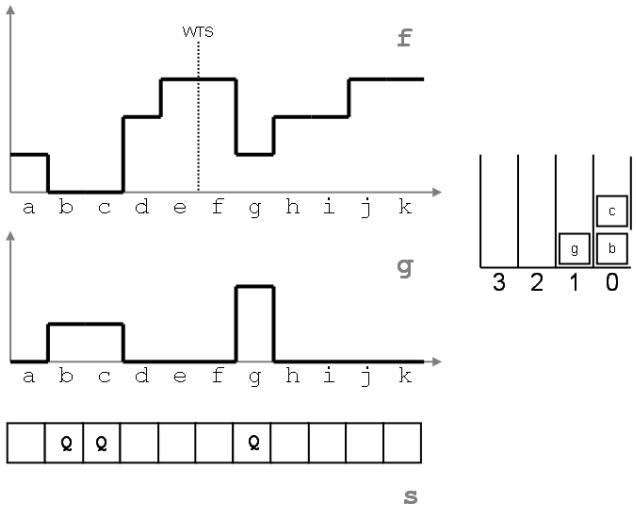
The HQ operation is controlled by the following algorithm (pseudo-code) applied to each token:

```
{
For each priority queue, from priority 0 to N, do:
    {
    While the queue is not empty, do:
        {
        - Extract a token x from the HQ
        - Determine the neighboring pixels of x which are CANDIDATE in s
        - For each neighbor y with a CANDIDATE tag, do:
        {
            - Assign to y in image g the same label as x.
            - Insert the token y into the queue with priority
            corresponding to the grey level of y in f (if it exists)
            or into the queue of highest priority still existing.
            - Mark y as QUEUED in the status image s.
        }
    }
}
```

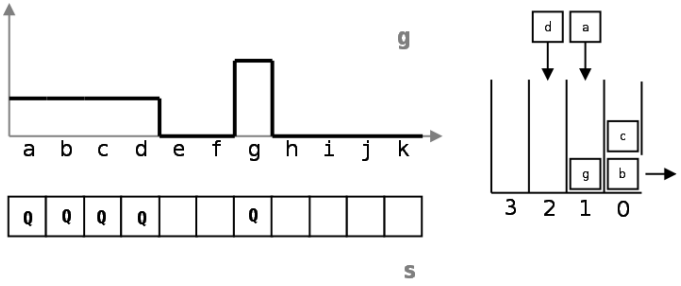
At the end of the process, each catchment basin is tagged with a strictly positive label corresponding to the label of the source. The label image thus indicates, during the process, which catchment basin each pixel belongs to.

3.1.3 Example and discussion

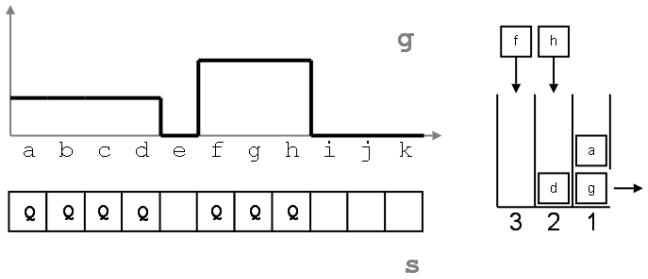
This operation is illustrated in a simple case (one-dimensional).



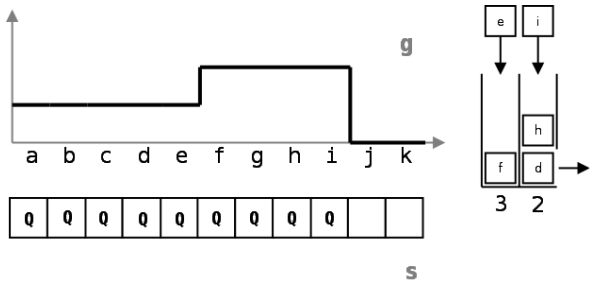
Initial state: in this example, the flooding sources correspond to the minima of f. Tokens (b), (c) and (g) are pushed on the HQ in their respective priority queues. Image g is initialised with two labels (1 and 2). The status image contains CANDIDATE tags except for source pixels which are marked QUEUED.



First step of the process: token (b) is extracted, its only CANDIDATE neighbor (a) takes the same label in g, is pushed on the HQ (priority 1) and is tagged QUEUED in s. Same treatment is applied to token (c). Priority 0 queue is then empty and is removed.



Second step: token (g) is processed, it neighbors (f) and (h) are placed on the HQ and tagged QUEUED. Token (a) has no CANDIDATE neighbor.



Next step: tokens (d) and (h) are extracted. Token (d) sends (e) on the queue. (h) pushes (i) on the current priority queue. So (i) is processed and pushes (j). The process ends up when (f), (j) and (k) are processed.

Let us recall the importance of the rule for managing disappeared queues. Sometimes, indeed, lower priority tokens appear when the corresponding stack no longer exists. In this case, the token is inserted into the current stack. This situation occurs frequently with marker-controlled WTS, but not only in this case. Figure 3 shows the use of a HQ when initialisation is performed with any kind of markers and not with minima of **f**. Indeed, in this case, tokens appear, of higher priority than the current queue. Everything happens as if the grey value of the corresponding pixel was raised to the value corresponding to the current height of the flooding, which can be considered, somehow, as a change of homotopy "on the fly" [2]. However, this rule is also essential in the case of the construction of a simple WTS where it can happen that points with a height less than the current flood have not been taken into account yet (especially in the buttonholes structures, frequently encountered in real images, see [2]).

3.1.4 Strengths and weaknesses

This implementation is quite simple and therefore it is very fast. In MAMBA, the original image **f** is always a 8-bit (greyscale) image. The label image **g** is a 24-bit image and the status image **s** a 8-bit image. In fact, **g** and **s** are sharing respectively the three lower bytes and the higher byte of a 32-bit image. The depth of **g** allows to use up to $2^{24} - 1$ markers (16,777,215) in the computation of the WTS. If a marker is assigned to each pixel of an image (a rather clumsy hypothesis by the way...), this means that the algorithm could manage images containing this number of pixels (typically 4096x4096 images for instance). Fortunately, in most applications, we need a much less number of markers.

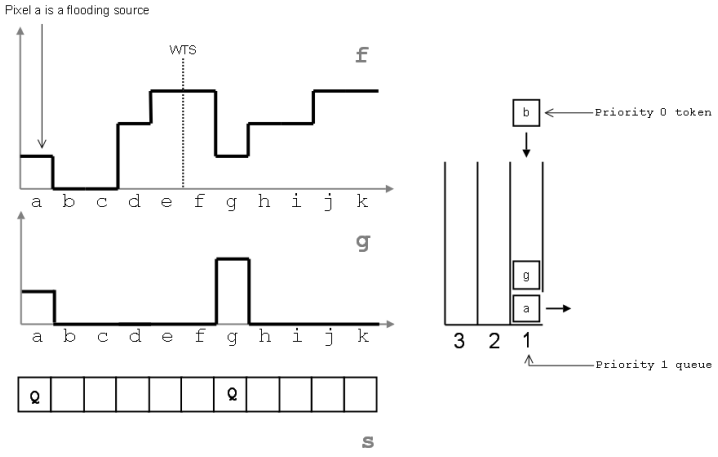


Figure 3: Management rule for the lower priority tokens

The computation time is proportional to the size of image **f** which must be a greyscale image. It is possible, however, to use this implementation to compute a watershed transformation on a 32-bit image (see section 6.1.1). remember also that this implementation is not isotropic as tokens queueing introduces a processing order which has important consequences on the result.

3.2 Watershed algorithm with real boundaries (watershed lines)

When catchment boundaries are produced, the watershed algorithm implementation is more complex. The status image and its tags are more sophisticated. The label image may also be more complex.

Note that, although this implementation has been widely used in past and recent libraries, to the best of our knowledge, no formal description of this algorithm exists in the literature. Therefore, this presentation will give us the opportunity to highlight some particularities in the initialisation process and in the algorithm operation.

3.2.1 Initialisation of the process

A label image **g** and a status image **s** are still used. The first one contains the catchment basins labels as previously and the other one tags indicating the status of image points. This status may be one of the following:

- "CANDIDATE", if the point is candidate to be inserted in the HQ.
- "QUEUED", if the point has already been inserted in the HQ.
- "RG_LABELLED", if the point received a label corresponding to a catchment basin.
- "WTS_LABELLED", if it belongs to the watershed line (WTS).

Compared to the previous algorithm, two more tags have been added. Their purpose is to specifically handle the status of the watershed points.

The label image contains strictly positive label values corresponding to the various catchment basins in progress and the value 0 corresponding to points not assigned yet. Associated with that image, there is a HQ.

The HQ and the status image initialisations are more complex than in the previous algorithm. The initialisation is performed by placing the initial marker points on the queue and by storing their corresponding labels in the label image. They are also tagged "RG_LABELLED" in the status image. When these tokens come out of the queue, they are, by definition, already marked "RG_LABELLED" and their neighbors can only have a "CANDIDATE", "WTS_LABELLED" (surprising, but possible ...) or "RG_LABELLED" status but with a label identical to the extracted point (in this version of the WTS with boundaries, original markers cannot be adjacent). All candidate neighbors must therefore be pushed on the HQ thus ensuring the spreading of the flood. The label of these points could possibly be replaced with a label... of same value! They will never become WTS points.

3.2.2 HQ processing, pseudo-code

Once initialised, the operation on this HQ is the following:

```
As long as the HQ is not empty, do:
{
    - Extract a token x from the HQ (from the active higher priority queue)
    - For each of its neighboring points:
    {
        - If the neighbor point is "CANDIDATE" , its coordinates
          are stored in a buffer. This buffer size is equal to
          the number of neighbor points (8 in square grid, 6 in
          hexagonal grid) and therefore contains potentially
          liable to flooding neighbors.
        - Otherwise, if the neighbor point is "RG_LABELLED", then:
        {
            - If the extracted pixel is "RG_LABELLED" and if its label
              is different from the label of the neighbor point, the
              extracted pixel is therefore a WTS point and it is
              marked as such ("WTS_LABELLED).
            - Otherwise, the extracted pixel is marked "RG_LABELLED"
              and it inherits the label of its neighbor in the label
              image.
        }
    }
    - If at the end, when all neighbors have been taken into account,
      the extracted point is marked "RG_LABELLED", the points of the
      buffer are inserted into the HQ and are marked "QUEUED"(so as
      not to be inserted more than once  on the HQ).
}
```

Let us explain briefly this algorithm. Any token leaving the queue should have its status determined definitely: it is either "RG_LABELLED" or "WTS_LABELLED". But this status is determined only when all its neighboring points have been examined. Depending on the final status, its "CANDIDATE" neighbors can then be pushed or not to the HQ: if the pixel is "RG_LABELLED", it is legitimate for its "CANDIDATE" neighbors to be pushed on the queue because they are adjacent to a flooded point and then, the flood will spread to these points. On the contrary, if the point belongs to a watershed line (it is "WTS_LABELLED"), it cannot propagate flooding as it is not flooded itself. Its "CANDIDATE" neighbors must remain "CANDIDATE". This is acknowledged by storing its "CANDIDATE" neighbors in a temporary buffer and by pushing them to the HQ once it is certain that the central point is "RG_LABELLED" and not "WTS_LABELLED".

Note that the order of examination of the neighbors is not imposed and it depends on the implementation. In the following examples, we shall suppose that this order is clockwise starting from the upper right neighbor.

Note also that, in this algorithm, all the neighbors are processed even when the status of the extracted token has been fully determined by a previous neighbor.

3.2.3 Example of processing

Here is an example of the algorithm applied to the image in Figure 4, defined on an hexagonal grid (the same algorithm applies to a square grid image without any problem).

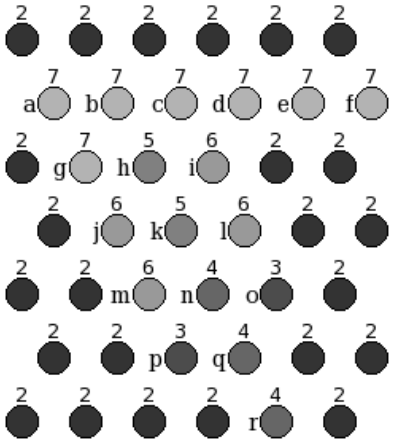


Figure 4: Original image used to illustrate the algorithm

The numerical values correspond to the value of each pixel (here the contrast is artificially increased). Pixels that are not markers (markers all have the value 2) are marked with a letter.

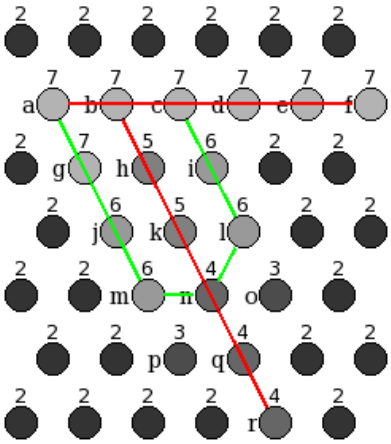
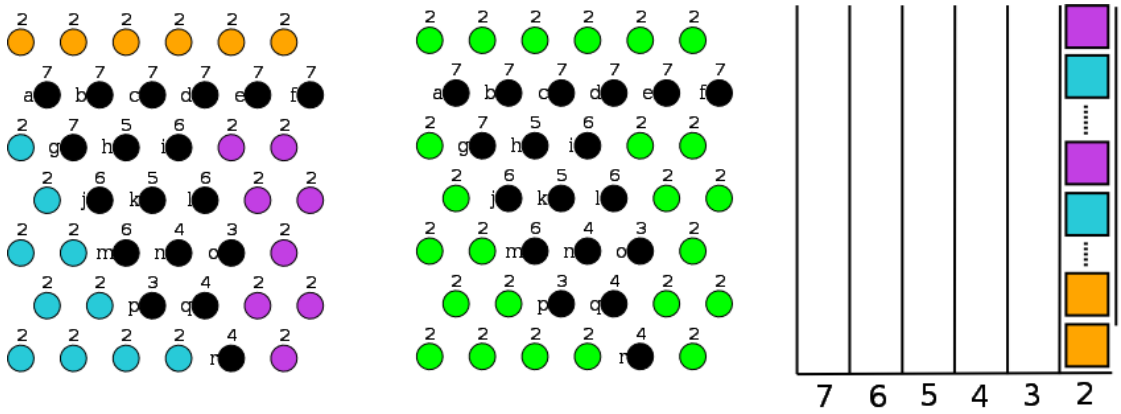
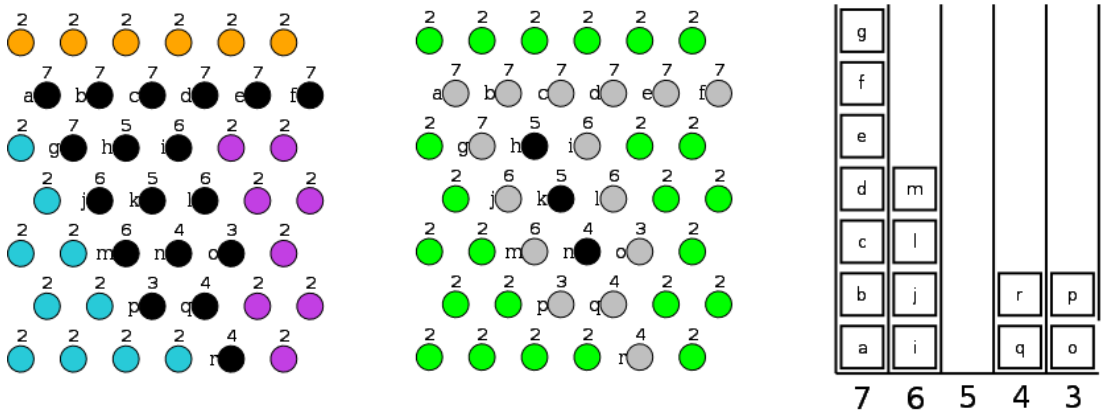


Figure 5: Button hole configuration

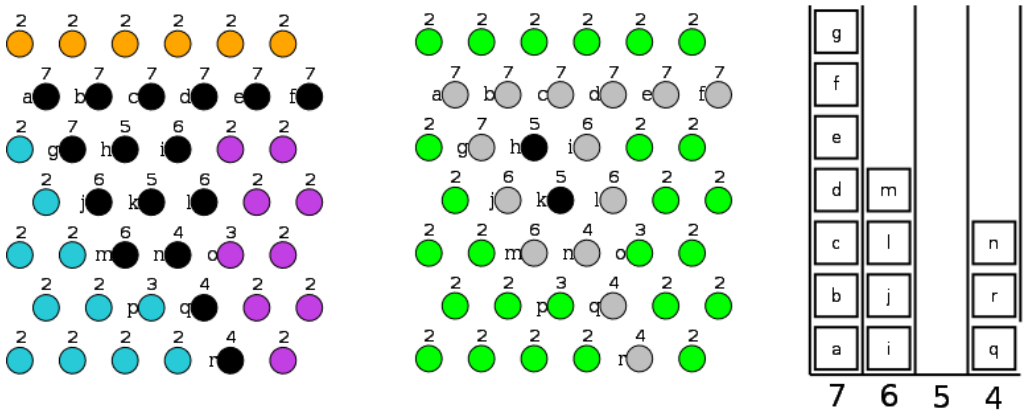
The image depicted in Figure 5 is a buttonhole (see [2]). Lips of the buttonhole are indicated in green. Its entry point is point (n). The awaited watershed lines at the end of the process are in red.



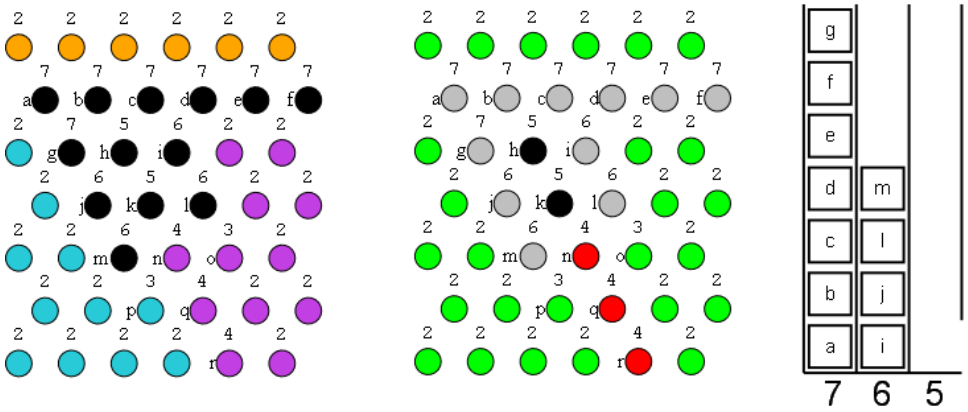
Initialisation step. At left, the label image with labels in orange, blue and purple. These pixels correspond to the initial three markers (the colors correspond to the values of these labels). These pixels are pushed on the HQ during initialisation. They are all pushed in priority 2 queue (they have not all been represented) and are "RG_LABELLED" in the status image (middle image, in green). The black dots in the status image are not yet included in the HQ ("CANDIDATE" points). At right, the corresponding status of the queue after initialisation.



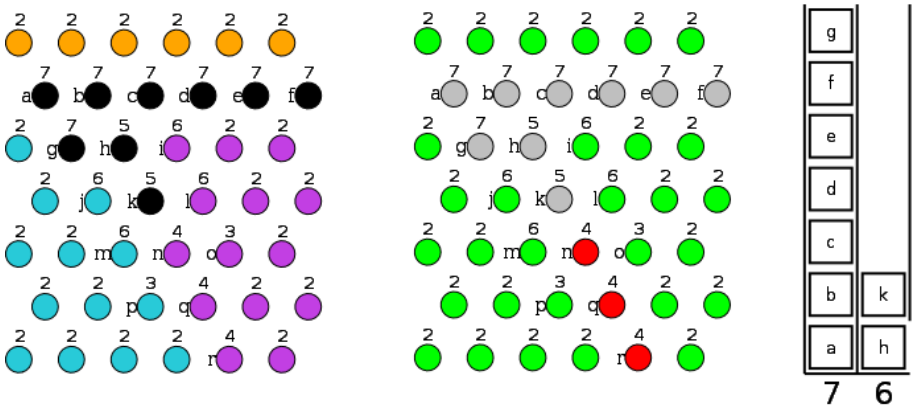
Treatment of priority 2 queue. Each pixel of the queue is surrounded either by pixels of same label or by "CANDIDATE" pixels. Therefore, each token in the priority 2 queue pushes, when extracted, its candidate neighbors on the HQ. These points are labelled "QUEUED" in the status image (in grey). The label image is not modified.



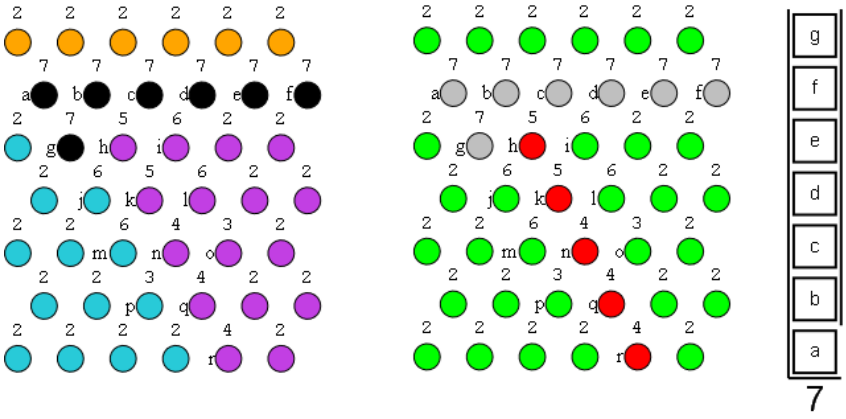
Situation when priority 3 queue has been entirely processed: tokens (o) and (p) were extracted. (o) found point (n) as a candidate ("CANDIDATE") to the inclusion in the HQ. This point has been labelled "QUEUED" (grey), inserted on the HQ in the priority queue 4 corresponding to its value and, during the checking of all its neighbors, the status of point (o) itself was modified. It took the purple label. Finally (p) was extracted, it produced no insertion of a neighboring point (no "CANDIDATE" black neighbor) and it took the blue label.



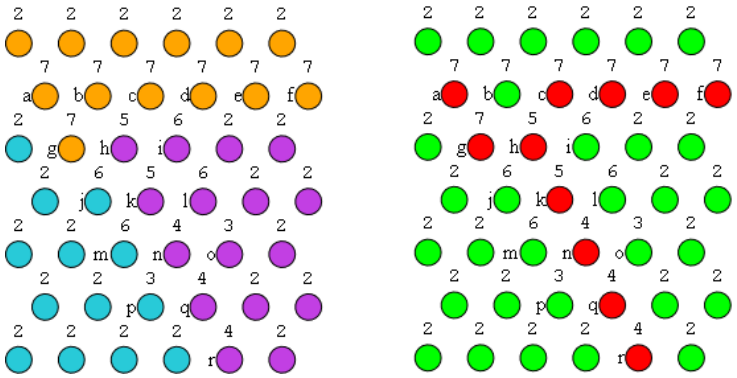
Treatment of priority 4 queue. Tokens (q) (r) and (n) are extracted from the queue. They become WTS points. However, although (n) has point (k) as a candidate, it is not included in the HQ because point (n) is not a point marked "RG_LABELLED" (it is not flooded). The queue processing ends (no points are inserted). The priority 5 queue is empty, the process continues with the priority 6 queue.



Treatment of priority 6 queue. Tokens (i), (j) (l) and (m) are extracted in this order. (i) is labelled (purple label) and, being labelled "RG_LABELLED", it pushes on the HQ tokens (h) and (k). The priority 5 queue being disappeared, (h) and (k) are placed on the current queue.



(h) and (k) are extracted from the queue and labelled as WTS points. The rest of the process is similar and will produce the same result as above.



Processing the last queue (priority 7). Points (a), (c), (d), (e) and (f) become WTS points. Point (b) has only two types of neighbors: WTS points and a single labelled point. It takes the label of this latter point (orange label). Note that the final result is different from the result expected above. It is however fully valid. The difference comes from the fact that points have not been processed in the same order in the HQ.

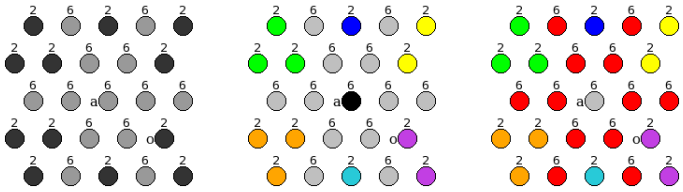
3.2.4 Strengths and weaknesses

This algorithm extends the principles established in the first algorithm (which computes only the basins) to allow the computation of the watershed lines. It remains fast (although not as fast as the first one) and straighforward.

Another important property of this implementation is that the WTS is idempotent (which was not true for some WTS implementations, with or whitout HQ, which can be found in some mathematical morphology software libraries).

Idempotence is a very important property of a watershed transformation. It means that, if we take the result of a watershed transform (that is the valued watershed function, see [2]) and if we apply again a watershed transformation on this function, the result is unchanged. This property is not always fulfilled in some watershed HQ implementations and a shift of the watershed lines may be observed. This defect is very annoying especially when the watershed transformation is used in hierarchical segmentations. However, we can guarantee that the watershed implementation in MAMBA is idempotent.

However, a remaining problem concerns points that are not labelled (because they are surrounded by WTS points). These points are never inserted in the HQ. Therefore, they are still labelled "CANDIDATE" at the end of the procedure. So, it is necessary to modifiy this tag (to change it to "WTS_LABELLED") to obtain the final WTS result. This is done through a simple look-up table applied on all the pixels of the status image. Note that, as, in some applications, the result of the watershed transform is not used as such but after a threshold to get the binary image of the watershed lines, this final cleaning could be merged with the thresholding through an appropriate look-up transform. However, for the sake of simplicity, this cleaning procedure has been kept in the MAMBA library.



Example of configuration inducing a "thick" WTS (top left). For the sake of simplicity, label and status images have been mixed (color conventions are the same). At right, the colored dots correspond to different catchment basins and associated labels. The grey points are the points "QUEUED" stored on the HQ. Point (a) is not yet inserted. The extraction of grey points stored on the queue will cause the insertion of point (a) in the current priority queue (priority 6) because it is "CANDIDATE". All extracted points will be considered as WTS points because they are all surrounded by points with at least two different labels. Point (a) is then removed from the queue and it is not labelled because it is only surrounded by WTS points.

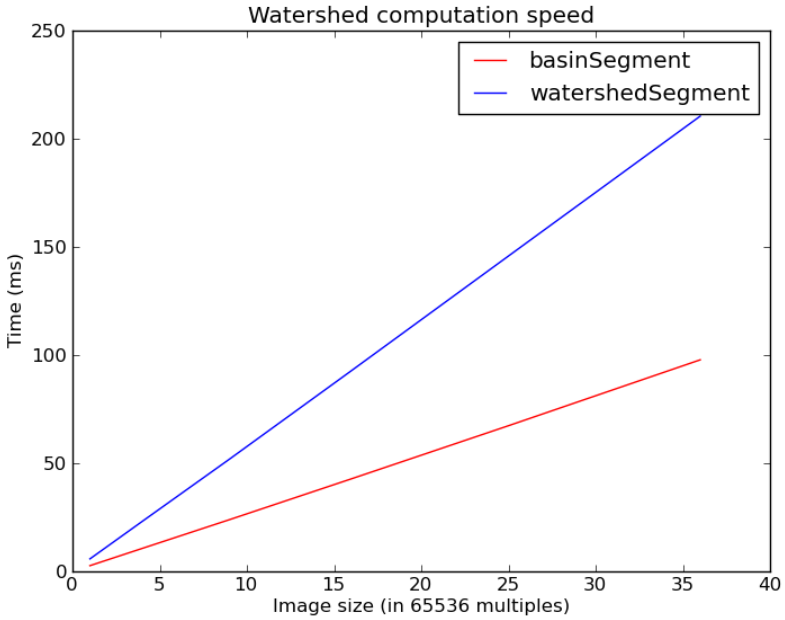


Figure 6: Computation time of the watershed transformation with HQ for various image sizes

3.3 Summary and discussion

Both watershed transform implementations in MAMBA (*basinSegment* and *watershedSegment*) are efficient and straightforward. The initialisation process is simple and the HQ operation is in accordance with the general description given in section 2. Moreover, the watershed transform in MAMBA is idempotent. It is also fast and its computation time is more or less proportional to the number of pixels in the image as it can be shown in Figure 6. These values have been obtained on a 2.66 GHz processor running on Windows XP with different sizes of images. Only the watershed or catchment basins computation times have been calculated. The markers labelling has not been taken into account. As expected, the *basinSegment* transformation is faster than the *watershedSegment* one.

4 Reconstruction algorithms

HQ can also be used efficiently to perform geodesic reconstructions by dilations or erosions. A geodesic reconstruction is obtained by successive geodesic dilations of an image **g**, the marker image, under an image **f**, the mask image, until idempotence is reached (no evolution of the transform). Using HQ to perform this transformation allows a dramatic reduction of its computation time. The dual reconstruction is obtained by achieving successive erosions of image **g** above image **f** until idempotence.

4.1 Dual reconstruction by HQ

Let us start with the dual reconstruction using a HQ, where the priority management in the queues is similar to the one which is used in the WTS operators. We refer again to [4] for the description of the algorithm where only this dual reconstruction (erosion-reconstruction) is described. Let us recall it before addressing the issue raised by the initialisation proposed in this description.

4.1.1 Initialisation

Dual reconstruction $R^*_f(g)$ of a function **f** by a function **g** with $\mathbf{g} \geq \mathbf{f}$, uses, as the WTS, a status image s. This status image (different from image **g**) will contain the various labels (or tags) indicating the status of the different points of image **g** during the reconstruction process. The label image **g** (also called marker image) contains the image to be rebuilt. This image will be modified and will hold the rebuilt image at the end of the reconstruction. Image **f** (also called mask image) remains unchanged. Three labels are possible for the points of the label image:

- "FINAL" indicates that the point has received its final value in the reconstruction.
- "QUEUED" indicates that the point has already been inserted in the HQ, but that it has not taken its final value yet.
- "CANDIDATE" means that the point has not been taken into account (this label corresponds normally to the value 0).

HQ handling in the geodesic reconstruction is very similar to its operation in the WTS, at least when the watershed lines are not drawn (catchment basins only). In particular, the priority of each queue is directly related to the grey values of image **g**, as it was the case in the WTS implementation. This means that pixels (and tokens) of lower grey value are processed first in this algorithm.

However, in the algorithm described in [4], the HQ initialisation presents a strange singularity. The HQ initialization is made by storing at least one point of each regional minimum of function **g**. But nothing is said about how to extract those points belonging to regional minima of **g**. This is all the more disturbing since the geodesic reconstruction is conventionally used to extract extrema (minima and maxima) of an image. With this initialisation, the geodesic reconstruction by HQ seems tautological: to realise it, we must first detect points belonging to the minima of image **g**. But, usually (and in Mamba in particular), these points are detected by an reconstruction operator! Everything is happening as if, to perform a reconstruction using a HQ, we were obliged to first perform a mimina extraction by means of a geodesic reconstruction realised by another algorithm, a recursive oerator for instance (this operator exists in MAMBA but it is slower than the reconstruction by HQ, see section 4.3).

Fortunately, this level of complexity is not necessary. Indeed, it is essential that at least one point belonging to each minimum of **g** be included in the initial set of tokens. But we can add to this set any number of extra points without changing in any way the final result. In particular, we can initiate the process with all the points of **g**, which eliminates the screening step described above.

This ensues from a well-known property of the reconstruction (direct or dual). You can replace image **g** by any image **g'** (**g'** $\geq$ **f**) provided that minima of **g** and **g'** are identical. One can even further relax the condition by considering functions **g** and **g'** such that the intersection of the connected components of their minima is pairwise non-empty (Figure 7).

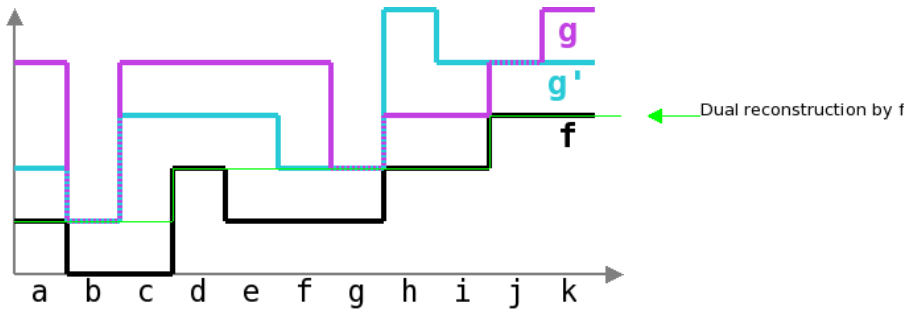


Figure 7: The two images g and g' provide identical dual reconstructions as their minima are superposed

Another important point is to ensure that $\mathbf{g} \geq \mathbf{f}$. As it is not always the case, this condition is enforced by using the image max(**g**(x), **f**(x)) in the initialisation phase which is therefore the following:

```
For any pixel x in the image:
{
    - Assign point x  a new value  in image g equal to:
            g(y) = max(g(x), f(x))
      The token corresponding to point x is then inserted
      into the priority queue g(x) and the point is
      labelled "CANDIDATE" in the label image.
}
```

4.1.2 HQ processing for dual reconstruction

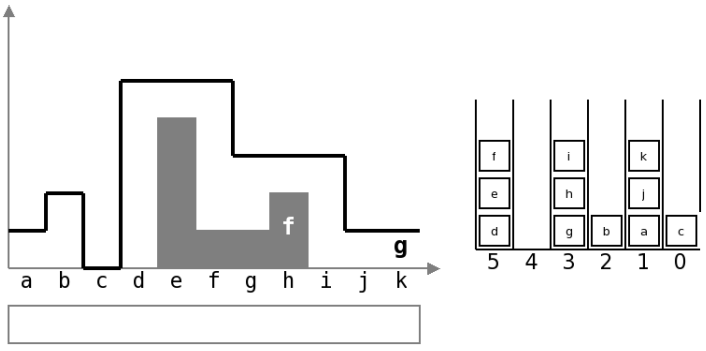
Once initialised, the hierarchical queue is processed according to the following algorithm:

```
While the HQ is not empty, do:
{
    - Retrieve a token x from the HQ (highest priority queue).
    {
        - If the label of x indicates that x does not already
          have its final value (label other than "FINAL"), give
          x the label "FINAL", indicating that its value is now final.
        - For every neighbor y of x with a "CANDIDATE" label
          (zero label), assign a new value at this point y in g
          image equal to:
                g(y) = max(g(x), f(y))
          The token corresponding to point y is then inserted into
          the g(y) priority queue and the point is labelled "QUEUED"
          in the label imagel.
    }
}
```

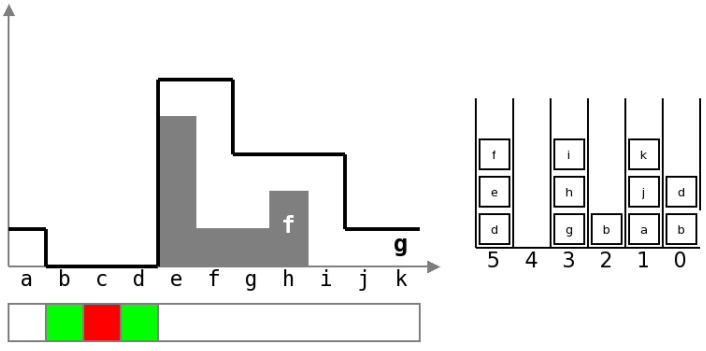
This algorithm consists simply in performing a geodesic erosion of each pixel of **g** by starting from the lowest ones and by propagating this erosion to neighboring pixels. This propagation is very fast as it avoids to take into account unnecessary pixels.

4.1.3 Example of dual reconstruction with a HQ

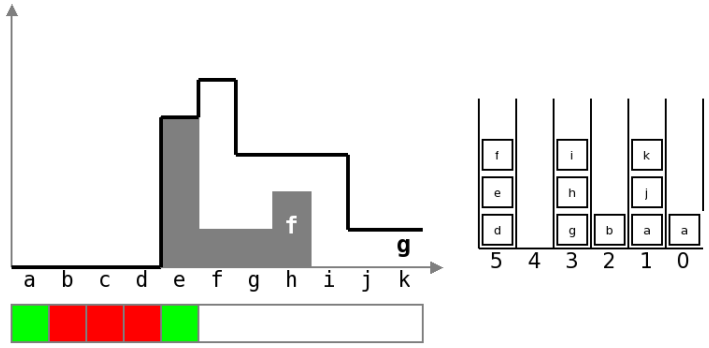
We will illustrate this approach by using again the one-dimensional example provided in [4]. Although this example is very simple, there is absolutely no difference in the HQ operation when 2D images are processed. We represent image **g** at each step (image **f**, in grey, is unchanged), the status image and the HQ.



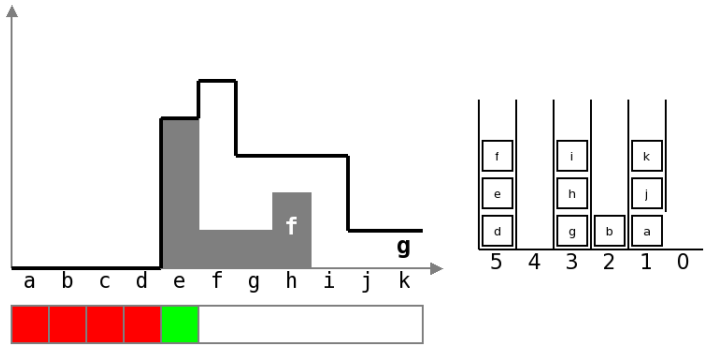
Initialisation: all points of g are stored on the HQ in the stack corresponding to their value. The status image only contains "CANDIDATE" labels (colored in white).



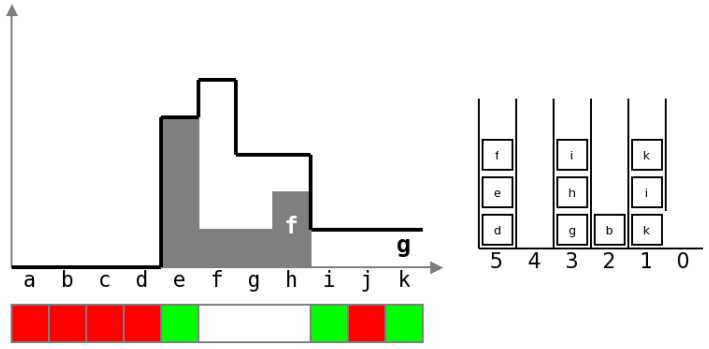
The first token (c) is removed from the queue of highest priority. It is marked "FINAL" (red) in the label image. The neighbor (b) of (c) is "CANDIDATE", it is stored in the HQ in the stack of priority max(g(c), f(b)) = 0, value also taken by g(b). The point (b) is marked "QUEUED" (green) in the label image. The same treatment is applied to point (d), neighbor of (c). We note that (b) and (d) are inserted in the queue whereas they were already present (on higher priority queues). We will see that this feature does not affect the result.



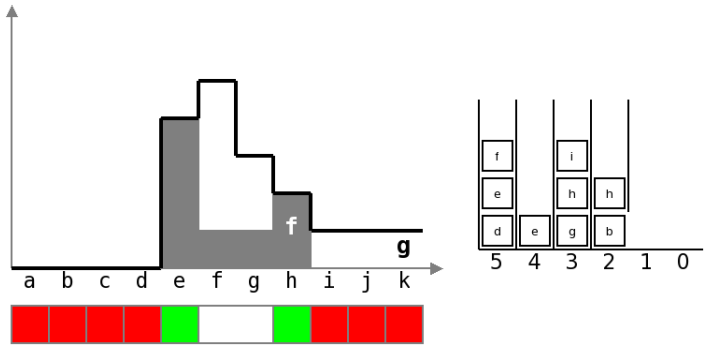
Token (b) exits from the queue, it is marked "FINAL". Point (a), "CANDIDATE" neighbor of (b) takes the value max(g(b), f(a)) = 0 and is inserted into the priority queue 0. It is marked "QUEUED" in the label image. Same treatment is applied for point (d), marked "FINAL" that pushes point (e) on the stack (marked "QUEUED") with priority g(e) = max(g(d), f(e)) = 4.



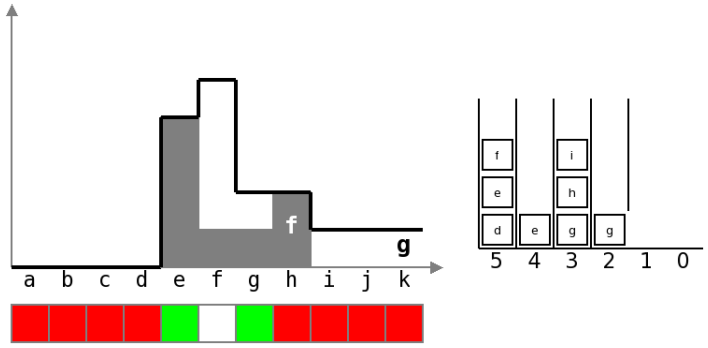
Token (a) leaves the stack (it is marked "FINAL"). Point (a) had no neighbors labelled "CANDIDATE", no points are inserted and the stack of priority 0 is empty.



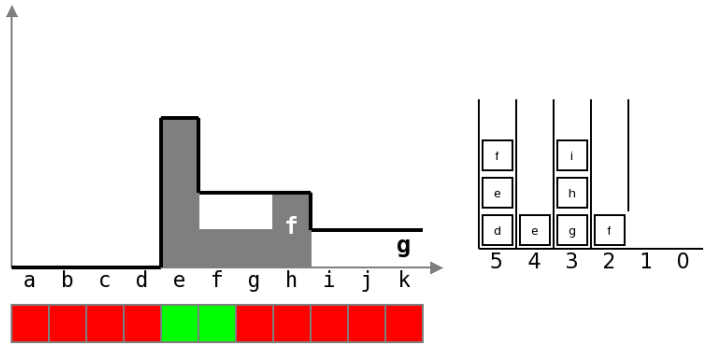
The priority 1 queue is empty. Point (a) exits, it is already marked "FINAL" and it has no "CANDIDATE" neighbor. It cannot have one since it has been processed already and its neighbors have therefore already been taken into account. This shows that to insert a point several times in the HQ has no adverse consequences because, when it is reinserted, it is always in a lesser or equal priority queue than the queue it was previously. Point (j) is extracted. It takes the "FINAL" label and it pushes ponts (i) and (k) on the stack with the same priority. The image g and the label image are modified accordingly. We see that the point (k) is reinserted into the queue with the same priority.



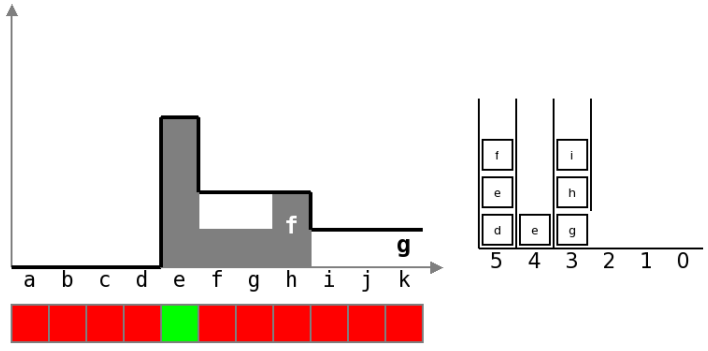
Token (k) is extracted. This token was inserted in the initialisation phase, but it does not matter. We can already guess that the second token corresponding to (k) and located in the same queue will simply be removed when its turn comes. (k) is marked "FINAL" and has no "CANDIDATE" neighbors. Point (i) is also marked "FINAL" and pushes its "CANDIDATE" neighbor (h) on the priority 2 ((h) is marked "QUEUED"). g(h) takes the value 2. The second occurrence of (k) is simply removed.



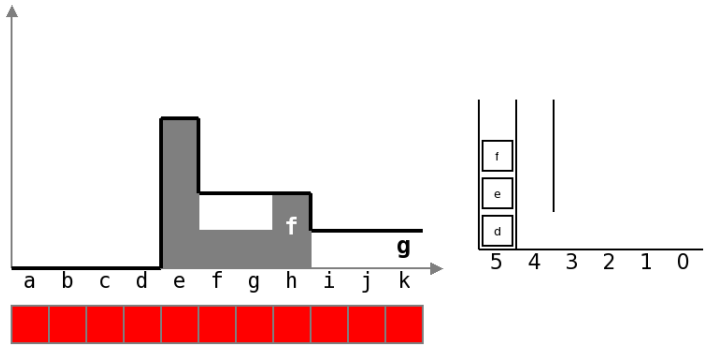
Token (b) is exited. Token (h) comes out and pushes its neighbor (g) on the current priority queue. g(g) is amended.



Point (g) is processed and pushes its neighbor (f) on the current priority queue. g (f) is modified.



Point (f) is processed and does not push any point in the HQ (no neighboring point is "CANDIDATE").



End of the process. Tokens (g), (h) and (i) are extracted. (e) is marked "FINAL". Finally, (d), (e) and (f) are removed from the priority queue 5.

4.1.4 Strengths and weaknesses

The previous example shows that there is no problem to initialise the HQ with all the points of the original image **g**. In fact, this approach amounts to generate a recursive process in the label image and in the HQ since tokens are reintroduced thus replacing the first insertions and somehow cancelling them. Note that this algorithm requires a HQ larger than the number of image points. This size can theoretically be up to twice the size of the image (if a single minimum is present, which could, associated to a very specific configuration, lead to the reinsertion of all other points). We will see below how the HQ has been designed in MAMBA in order to simplify its management and to cope with this problem.

Due to the insertion, another consequence is that the processing time is not proportional to the number of pixels in the image, although the reinserted pixels processing is very fast in the second pass (since it is sufficient to unstack them). We could reduce the number of reinsertions and, thereby, reduce the HQ size and processing time by initialising the process with a reduced set of tokens as already explained above. However, this always needs some pre-processing (in order to generate this reduced set) which at the end decreases the performances of the overall process.

4.2 Reconstruction by HQ

The direct reconstruction $R_f(g)$ by dilation can also be performed with a HQ, mutatis mutandis. Indeed, the reconstruction $R_f(g)$ of image **g** under **f** is equal to $R_f(g) = m - (R^*_{m-f}(m-g)$ where m can be any revolving value (see [3]). If m is equal to 255, (m - f) and (m - g) represent the inverted images. Therefore, the algorithm design for the dual reconstruction must be modified accordingly.

4.2.1 Initialisation

The initialisation step is identical for the reconstruction to the initialisation realised for the dual reconstruction. The HQ is loaded with tokens representing the pixels of min(**g**, **f**) to ensure that $\mathbf{g} \leq \mathbf{f}$. The only modification lies on the priority of the queues. The priority order must be reverted. Pixels of image **g** of highest values will be processed first. This means that the priority 0 queue (highest priority queue considering the conventional priority numbering) contains tokens corresponding to pixels with a grey value equal to 255.

4.2.2 HQ operation

The pseudo-code describing the HQ operation is also almost identical. Queues processing starts with highest priority queues which now contain tokens of highest grey values. The rule for changing **g** values becomes:

```
new value of g(y) = min(g(x),f(y))
```

This corresponds to the propagation of a geodesic dilation starting from the highest values of **g**.

4.3 Discussion: reconstruction by HQ versus its directional implementation

MAMBA offers two possibilities to perform a reconstruction (or its dual operation) on a 8-bit image, the directional reconstruction (*build* function in *mambaComposed.geodesy.py* module) based on the *buildNeighbor* operator and the hierarchical reconstruction (*hierarBuild* function in *mamba.py* module). The first version has the advantage of supporting all MAMBA image depths. It is built with optimised operators (*buildNeighbor* is

a wrapper of *MB_BldNb8* that uses SSE2 instructions). Its main drawback is that its processing time is not proportional to the image size but to the complexity of the reconstruction. The HQ implementation does not use SSE2 instructions or specific optimizations but its characteristics make its running time almost proportional to the number of pixels in the image and not to the complexity of its reconstruction.

An example of complex image to be reconstructed is given in Figure 8. A single white dot at top left of the image is chosen as starting point for the reconstruction. The higher the number of zigzags, the higher the complexity of the reconstruction.

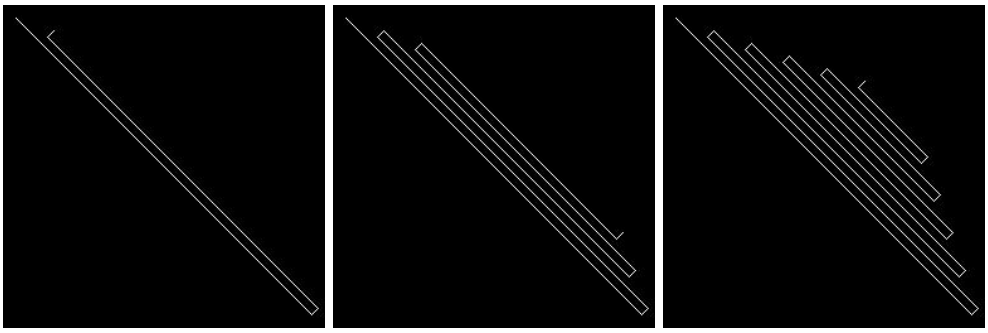


Figure 8: Examples of more and more complex reconstructions

The performances of such a reconstruction are given in Figure 9. The horizontal axis represents the number of loops in the image.

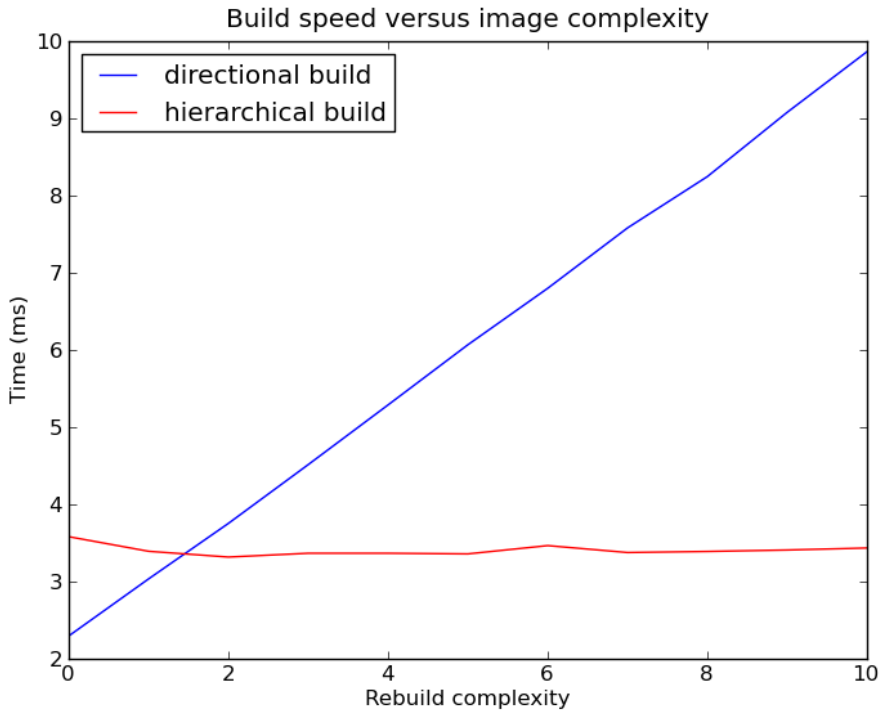


Figure 9: Reconstruction speed versus complexity

As expected, hierarchical reconstruction is stable regardless of the complexity of the image. Directional reconstruction is in turn penalised by the number of zigzags.

If this result validates the theory, however, it is hardly representative of a real situation. To compare the two approaches in a realistic situation, we conducted performance tests on image minima detection using dual reconstructions. The results are shown in Figure 10. The control parameter is the depth of the minima.

These results are clearly to the benefit of the HQ-based algorithm.

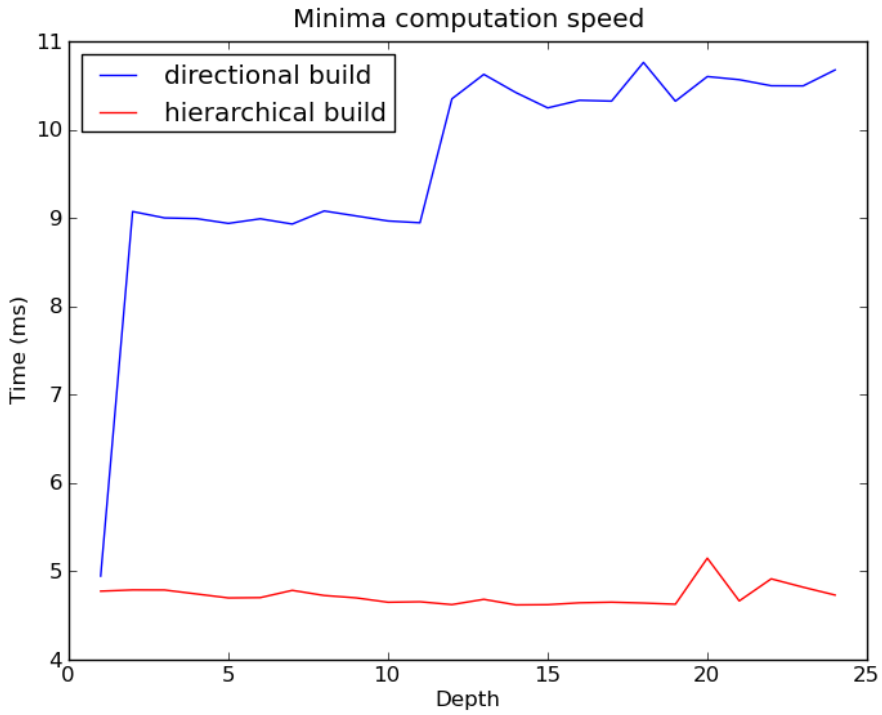


Figure 10: Computation time for the minima extraction versus their depth

5 MAMBA HQ implementation

The implementation of HQ algorithms in MAMBA is structured around tokens (*MB_Token* structure in the C code). A token is a simple structure which points only to the next token, thus allowing to chain them together in threaded lists. A list then corresponds to a queue and becomes a simple structure indicating the first and last tokens of the queue (thus allowing to stack another token on its top). Tokens are created in a 2-dimensional array of the same size as the processed image which allows to link a token with its corresponding pixel through its position in the tokens array (Figure 11).

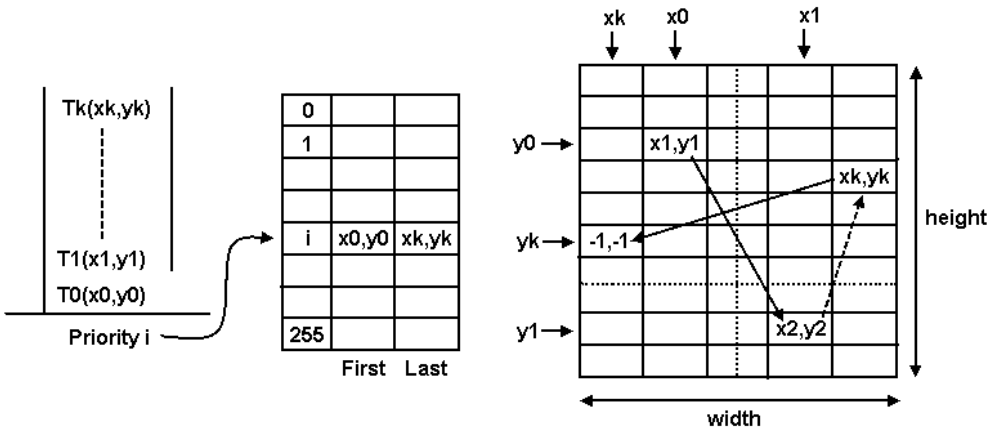


Figure 11: Hierarchical queue structure. Each queue has an entry in a hierarchical list indicating the first and last token stored in the queue. The token array is a threaded list which allows to scan the entire queue. Coordinates (-1, -1) for the next token indicates the end of the queue

The hierarchical queues algorithms were implemented in MAMBA for watershed transformations with two different functions. The first one, *basinSegment*, performs the watershed transformation without generating the

watershed lines. The corresponding C Implementation (*MB_Basins* function) is the simplest of those using a HQ in MAMBA. The second function, *watershedSegment*, generates the catchment basins and the watershed lines (basins boundaries). Its C implementation (*MB_Watershed* function) is rather complex to ensure its idempotence. For both algorithms, the number of tokens stacked in the queue cannot exceed the number of pixels in the image. Moreover, there is at any time a single occurrence of a given pixel, i.e. of its corresponding token in the HQ.

Regarding the implementations of geodesic reconstructions by HQ in MAMBA, two functions were created, *hierarBuild* and *hierarDualBuild*, which perform respectively a reconstruction and its dual operation. C implementations (*MB_HierarBld* and *MB_HierarDualBld* functions) are relatively simple, although they introduce some tricks to solve the specific problems coming with these reconstruction algorithms by HQ (such as initialisation with all image points). In this case, some pixels can be introduced a second time in the HQ even though the initial introduction has not been processed. So we sometimes need two tokens per pixel. To be able to safely use these two sets of tokens, the tokens array is composed of two parts. The first one is dedicated to the HQ initialisation, whilst the queue processing and the neighbors tokens insertion are using the second part of the tokens array (Figure 12). During treatment, it is possible to have two tokens corresponding to the same pixel in the queue. We must therefore control, for each token extracted from the queue, that its associated pixel has not already been processed. Another feature of the reconstruction by HQ is that the result is a 8-bit image. So, contrary to the watershed transform where the label image and the result were stored in different byte planes of a single 32-bit image, we cannot mix the result and the label image in reconstruction by HQ algorithms. Therefore, the label image is defined in a separate table.

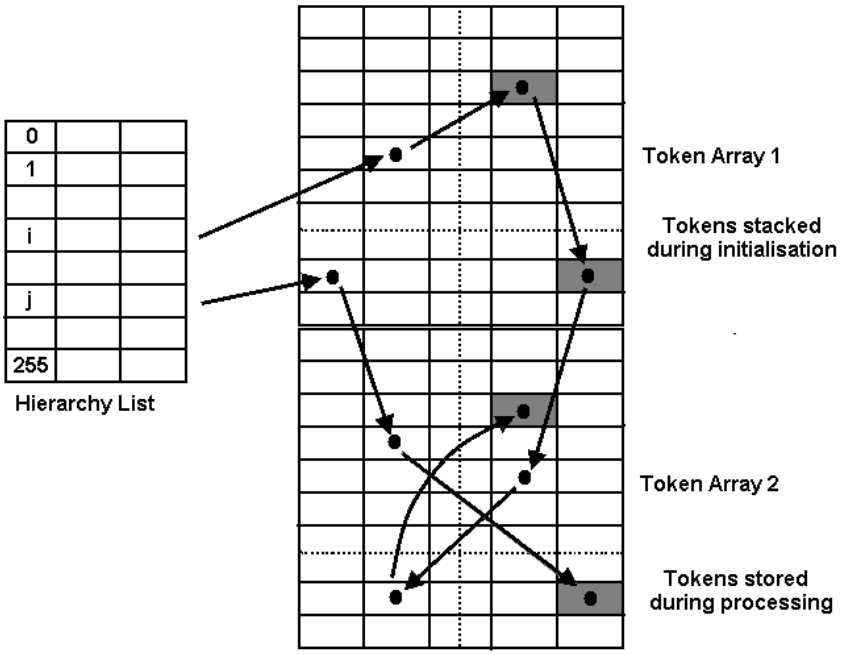


Figure 12: Double hierarchical queue implementation for the geodesic reconstruction. The grey tokens correspond to the same pixel (same position in the two token arrays) and have been stored twice in the HQ

6 Improvements and extensions of HQ

This section is devoted to the description of some extensions and improvements which can be applied to the HQ operations described previously to address some specific domains. Among them, we shall focus our interest on 32-bit images and 3D images. Regarding 32-bit images, extensions of HQ watershed transforms and HQ geodesic reconstructions will be described. Concerning 3D images, only general ideas aiming at extending HQ structures to this class of images will be given.

6.1 Hierarchical queues for 32-bit images

Using the HQ algorithms on 32-bit images is at first challenged by a problem of memory requirement. Indeed, on 8-bit images, the range of pixel values is 0 to 255. Creating a HQ of 256 queues presents no difficulty on any modern computer or even handheld devices (smartphone,...). 32-bit images however can potentially have pixel values range between 0 and $2^{32} - 1$. Creating a HQ with more than 4 billions queues is simply impossible.

The solution is to split the processing of 32-bit images slice by slice using efficiently the HQ structure designed for 8-bit images. 32-bit images are cut into different slices of equal thickness. Each slice is then processed in the HQ structure designed for 8-bit images. Finally, the results of the processing of these different slices are stacked again to produce the expected transformation on the initial 32-bit image. Obviously, some appropriate rules must be enforced to obtain unbiased results. The computation time of the transformations is generally proportional to the range of values found in the 32-bit images. This time is not much longer than the computation time awaited with a direct 32-bit implementation because this implementation would also require to operate a hierarchical list proportional to this range.

Of course, this approach could be directly implemented into a low-level C function that would directly process 32-bit images thus simplifying the process. This would also allow more flexibility and speed (for example the thickness of the slices could be increase to 512, 1024,... instead of the "forced" 256).

6.1.1 Watershed transforms for 32-bit images

Let us describe first the extension to 32-bit images of the watershed transform where only catchment basins are generated. This algorithm has been made possible thanks to the fact that it is possible, in MAMBA watershed implementation, to stop the flooding process at any altitude.

Figure 13 illustrates the principle of the algorithm. The original image is split into successive slices. The first slice contains all the pixels with a value in the range [0, 255]. Pixels with values greater than 255 are also given the value 255. The catchment basins of this function are built but the flooding is stopped at level 254. Then a second slice is defined, comprising pixel values in the range [254, 509]. These values are shifted in the range [0, 255]. Pixel values higher than 509 take the value 255. The catchment basins of this new function are built up to the level 254. The flooding sources are simply the catchment basins produced at the previous step. The procedure is repeated until the maximum value of the initial 32-bit image has been treated (that is, contained in a slice).

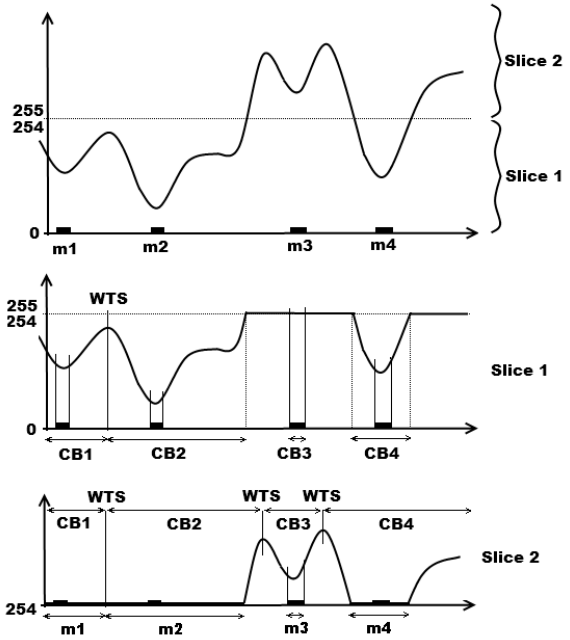


Figure 13: Principle of realisation of a watershed transform by slicing of a 32-bit image with a 8-bit HQ

Note the importance of stopping the flooding process at level 254. Indeed, if the flooding was continued to level 255, it would propagate on the flat zones added at this level. Therefore, the catchment basins would be adjacent and the entire process would end at the first step. Moreover, these catchment basins would be deeply

biased as the topography above this altitude would not be taken into account.

A similar approach can be used to compute the watershed lines of a 32-bit image. The only difference is in the definition of the sources of flooding when computing the watershed transformation of slice i. To get them, the watershed lines obtained at the end of step i-1 are extracted (note that they are not necessarily closed) from the status image. The corresponding pixels are "WTS_LABELLED". Then, these lines are subtracted from the label image. Therefore, this label image contains only the interior of the current catchment basins and the source pixels which have not started to pour water yet (since their altitude is higher than the current level). This modified label image is used as the new label image for the watershed transform at step i. At the end of the operation, watershed lines which have been removed at the beginning of this step are now regenerated again together with new watershed lines produced during the flooding of the pixels belonging to slice i. When all the slices have been processed, the status image contains the final watershed transformation of the initial 32-bit image.

6.1.2 Geodesic reconstructions for 32-bit images

This slicing technique can also be used to realise geodesic reconstructions by HQ on 32-bit images, as illustrated in Figure 14. The approach is even simpler than for the watershed transformation. We just have to cut the mask image and the marker image into slices, to replace the pixel values under the floor or above the ceiling of each slice by 0 and 255 respectively and to perform the reconstruction of the new mask function by the new marker one using the HQ algorithm. Then, the successive reconstructions just have to be added to get the final result.

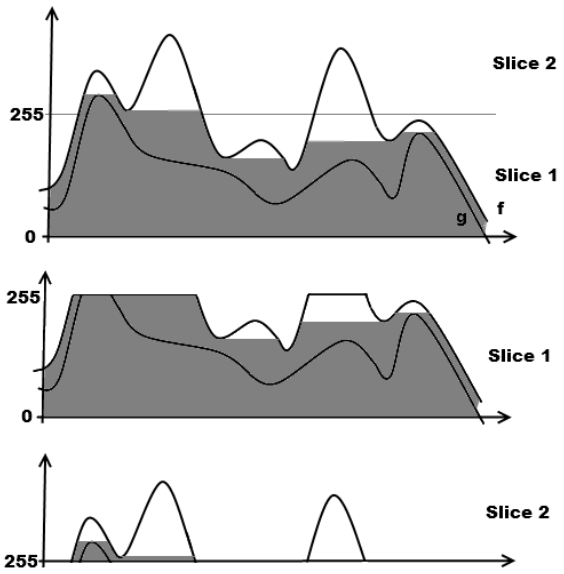


Figure 14: Reconstruction operators by slicing on 32-bit images

Note that this approach can be used equally for the geodesic reconstruction and for the dual reconstruction. Note also that it is not necessary to process the different slices in a specific order contrary to what is done with the watershed transformation. This is due to the fact that, in geodesic reconstructions, propagations are horizontal, whereas they are upstream in the watershed transform.

Remind that there exists already general reconstruction operators in MAMBA (build and dualBuild) which are based on a recursive algorithm and which can handle 32-bit images without any problem. Therefore, it is not sure that the algorithm described above will be faster than the recursive one for 32-bit images. It will depend on the complexity of the images, as already discussed in section 4.3, and also on the range of pixel values: the larger this range, the greater the number of slices.

6.2 Hierarchical queue for 3D images

Contrary to what was possible for 32-bit images, you cannot use the watershed and build operators present in MAMBA on 3D images. Indeed these types of images can be seen as sequences of 2D images but because the coded operators completely ignore the neighbors in the previous and the next image the result cannot be correct.

However the algorithms descriptions are still correct, only the definition of neighbor pixels is extended (and as for 2D images, depends on the grid used). Thus the same algorithms could be applied using specifically coded low-level C functions.

7 Conclusion

We tried in this paper to describe in a fairly exhaustive way both the principle of HQ operations and their implementations in the MAMBA image library.

The WTS and reconstructions implementations made in the MAMBA library brought original solutions for assessing the efficiency, accuracy and speed of the algorithms.

Regarding the watershed transformation (with apparent watershed lines), the initialization phase is simple and does not require pre-processing steps. In addition, the management of the neighbors of a token is made via a buffer memory, which reduces bias and increases processing speed by reducing breaks in the data stream. The final post-treatment could certainly be avoided. However, it is not very penalising and does not increase significantly the processing time.

The HQ reconstruction has also been greatly improved by simplifying the initialization step which now consists simply in filling the HQ with all pixels of the image to be reconstructed. The initialization as it was succinctly presented in previous documents describing this reconstruction was indeed rather strange since it required features (the extrema) that are generally obtained by means of algorithms that use or are very similar to reconstructions. The queue structure designed for watershed transforms is also used in the HQ reconstruction. Its duplication allows to assign more than one token to the same pixel.

The implementation of the hierarchical queue in MAMBA was performed using a threaded list built in the form of a two-dimensional array of tokens which allows a one-to-one correspondence between each token and the pixel that it represents. This structure opens the way for new uses of the HQ for increasing the performances of other morphological transformations.

References

[1] BEUCHER Nicolas, *MAMBA documentation.(mambaApi Reference Manual)*. Web documents (available at http://www.mamba-image.org), 2009-2011.

[2] BEUCHER Serge, *Segmentation Tools in Mathematical Morphology*. Invited conference, 12th International Congress of Stereology (ICS XII), Saint-Etienne, France, September 2007 (available at http://cmm.ensmp.fr/~beucher/publi/Course_ICS_print.pdf).

[3] BEUCHER Serge, *Geodesy and geodesic transformations*. Lecture slides, Summer school of Mathematical Morphology, Fontainebleau, France, September 2008 (available at http://cmm.ensmp.fr/~beucher/publi/Course2008_Geodesy_SB_eng.pdf).

[4] BEUCHER Serge, MEYER Fernand, *The Morphological approach of segmentation: the watershed transformation*. In Mathematical Morphology in Image Processing, (Dougherty E. editor) , Marcel Dekker, New York, 1992, http://cmm.ensmp.fr/~beucher/publi/SB_watershed.pdf.

[5] GRATIN Christophe, MEYER Fernand, *Mathematical morphology in 3-D*. 8th ICS, Irvine, 1991, in Acta Stereol. 1992, Vol. 11, Suppl. 1, pp. 551-558.

[6] MEYER Fernand, *Procédé de traitement d'images par files d'attente hiérarchisées*. French Patent n ° 91/033841, Institut National de la Propriété Industrielle, 20 mars 1991 (original document and its english translation can be found in the esp@cenet database[1])

[7] RIVEST Jean-François, GRATIN Christophe, TALBOT Hughes, *Xlim3D, an image analysis tool*. Published by Ecole Nationale Supérieure des Mines de Paris, Centre de Morphologie Mathmatique, 1994.

[8] MORPH-M, *Image Processing Software Specialized in Mathematical Morphology*. Web documentation (refer to the MORPH-M project Web page at http://cmm.ensmp.fr/Morph-M/index_en.html).

[1] This patent and all its international versions are lapsed since November 26, 2008.